1 formal Languages – 1

1. Introduction

Logic provides tools for systematically appraising reasoning. Central among these tools are formal languages.

The word 'formal' has a number of uses, at least two of which figure in logic. One use pertains to the word 'form', as used in contrast to 'content' or 'matter'. This is related to the crucial idea, which traces to Aristotle, that whether an argument is valid is a function of its form. The other use of the word 'formal' is in contrast to 'informal'. A related notion is the notion of rigor – formal languages are rigorous in the sense that they are stiff (think of the term 'rigor mortis') . Related to the formal/informal distinction is the distinction between natural language and artificial language; formal languages are a species of artificial language.

Leaving lexicography aside for the moment, 20th Century logic offers a concept of formal that is somewhat more specific. According to this concept, a formal language is a language that is *computationally recognizable* ("computable" for short). In other words, for a language to be formal in this sense, it must be possible *in principle* to program a computer so that it can distinguish between items in that language and items not in that language. It seems fairly clear that, if a language is formal in this sense, it is also formal in the more conventional sense.

2. The Basic Ingredients of a Formal Language

In order to specify a formal language $\mathcal{L}$, one does two things, *at a minimum*.

(1) one specifies the underlying set of *symbols* on which $\mathcal{L}$ is built;
(2) one specifies which strings of symbols count as *well-formed* in $\mathcal{L}$.

This is often described by saying that, in specifying a formal language $\mathcal{L}$, one specifies

(1) the *vocabulary* of $\mathcal{L}$
(2) the *rules of formation* of $\mathcal{L}$

Note: the term 'symbol' may be replaced by the term 'character', as used in many computer languages, such as Basic, Pascal, and C++. Notice in this connection that the term 'string' is also used in many computer languages, usually to mean string of characters.

Let us not worry too much about *exactly* what a symbol (character) is; for example, how exactly can we tell when we have a symbol rather than a natural number, a triangle, or a potato? Intuitively, symbols are the *fundamental units of writing* – the sorts of things one can write down on paper, key into a computer, inscribe in stone, spray-paint on a building, etc.

You are looking at lots of examples of symbols as you read this. You are also looking at lots of examples of strings of symbols. A string of symbols is just a (finite) bunch of symbols strung together in a row [notice that line breaks are ignored]. Most natural languages are formulated in this purely linear fashion. On the other hand, mathematics provides many examples of formulas that are not linear, but two-dimensional. Generally, however, there are systematic ways to "flatten" these two-dimensional formulas. In any case, we will concentrate on linear (flat) languages.

Some strings of symbols are grammatically significant (well-formed), and some are not. The text you are reading is full of examples of both. Pick two locations in this text; the literal material lying

between these two points is a string of symbols, but it need not (in isolation) be a grammatically significant unit; it need not be well-formed.

In summary (where $\mathcal{L}$ is a formal language):

> The vocabulary of $\mathcal{L}$ is the set of *symbols* employed by $\mathcal{L}$.

> The **rules of formation** of $\mathcal{L}$ specify which *strings of symbols* count as well-formed.

3. A Simple Example: The Language of Sentential Logic

The student of elementary logic is already familiar with a number of formal languages, including the language of sentential logic (SL). The formal presentation of the syntax of this language, $\mathcal{L}_S$, may be given as follows.

The language of SL – $\mathcal{L}_S$:

Vocabulary:

(1) upper case Roman letters:
A B C ……
(2) special symbols:
$\sim$ & $\vee$ $\rightarrow$ $\leftrightarrow$ ( )
(3) nothing else.

Rules of Formation:

(1) every upper case Roman letter is a formula;
(2) if $\mathbb{S}$ is a formula, then so is:
$\sim\mathbb{S}$
(3) if $\mathbb{S}_1$ and $\mathbb{S}_2$ are formulas, then so are:
$(\mathbb{S}_1 \,\&\, \mathbb{S}_2)$
$(\mathbb{S}_1 \vee \mathbb{S}_2)$
$(\mathbb{S}_1 \rightarrow \mathbb{S}_2)$
$(\mathbb{S}_1 \leftrightarrow \mathbb{S}_2)$
(4) nothing else is a formula.

Here, the block letter '$\mathbb{S}$' is used in order to suggest 'string'. The role of funny-looking letters will be explained shortly.

4. Another Simple Example of a Formal Language

In order to emphasize that the term 'language' in 'formal language' should not be taken too literally, we consider another example of a formal language, which we call $\mathcal{L}_2$.

Vocabulary:

(1) upper case Roman letters:
A B C ……
(2) nothing else is a symbol of $\mathcal{L}_2$.

Rules of Formation:

(1) if a string $\mathbb{S}$ begins with 'P', then it is a formula.
(2) nothing else is a formula of $\mathcal{L}_2$.

Admittedly, $\mathcal{L}_2$ is a silly formal language, with no obvious use in logic. Nevertheless, it is a formal language by our official criteria.

5. Back to SL

Recall that, as it is officially presented in Section 4, there are exactly 26 simple (atomic) formulas in the language of SL. Although this particular formal language is entirely adequate for all sensible exercises in elementary logic, it is not entirely adequate for all theoretical uses of sentential logic. The problem is that there are not enough atomic formulas.

The usual *informal* way to rectify this problem is to add numerical subscripts to the letters, so as to yield infinitely many atomic formulas. So in addition to 'P', 'Q', etc., we have 'P_1', 'Q_2', etc. This produces a new theoretical problem, however – how to formalize our rules of formation so that we can program our hypothetical computer to recognize formulas in the expanded language.

An alternative technique of generating infinitely many atomic formulas proceeds as follows.

Vocabulary:

(1) upper case Roman letters:
A B C ……
(2) special symbols:
♯ ~ & ∨ → ↔ ( )
(3) nothing else.

Rules of Formation:

(1) every upper case Roman letter is an atomic formula;
(2) if $\mathbb{S}$ is an atomic formula, then so is $\mathbb{S}$♯;
(3) nothing else is an atomic formula.

(1) every atomic formula is a formula
(2) if $\mathbb{S}$ is a formula, then so is:
$\sim\mathbb{S}$
(3) if $\mathbb{S}_1$ and $\mathbb{S}_2$ are formulas, then so are:
$(\mathbb{S}_1 \,\&\, \mathbb{S}_2)$
$(\mathbb{S}_1 \vee \mathbb{S}_2)$
$(\mathbb{S}_1 \rightarrow \mathbb{S}_2)$
$(\mathbb{S}_1 \leftrightarrow \mathbb{S}_2)$
(4) nothing else is a formula.

Notice that this formulation of the language of SL provides a precise way of rendering the numerical subscripts, suggested by the following examples.

P_1 : P♯
Q_2 : Q♯♯
R_3 : R♯♯♯

6. Object Language versus MetaLanguage

Before continuing, it is a good idea to discuss in some detail two very important related distinctions. First, there is the distinction between the *object language* and the *metalanguage*. Second, there is the affiliated distinction between *use* and *mention*.

First, the distinction between object language and metalanguage arises as soon as language is used to describe language. Consider French; there are two types of French grammar books — those written in French, and those written in a "foreign" language (say, English). In both cases, French is the subject of discussion (or object of discussion). However, whereas the former book is also written in French, the latter book is written in English. In both cases, French is the language under discussion – the *object language*. What about the language of use – the metalanguage? Well, in the former case, French is also the language of use (the *metalanguage*); in the latter case, English is the language of use.

This is summarized in the following table.

	French Grammar Book Written in French	French Grammar Book Written in English
Object Language	French	French
Meta Language	French	English

In the above example, the object language is a natural language. The object languages pertinent to logic are artificial (formal) languages. The metalanguage is (for us at least) a special dialect of English, which we will call English+ (English-Plus), which is ordinary English augmented with formal-syntactic machinery suitable for doing metalogic.

Even the presentation of elementary logic makes extensive use of the metalanguage English+. For example, the sentence

if $\mathbb{S}$ is a formula of SL, then so is $\sim\mathbb{S}$

is not itself a formula/sentence of SL. Rather, it is a formula/sentence of English+, which is the metalanguage used to describe SL.

Similarly, the rules of derivation are stated in the metalanguage. The gaudy letters ('$\mathcal{A}$', '$\mathcal{B}$', '$\mathbb{A}$', '$\mathbb{B}$', etc.) are not symbols of the object languages, but are rather part of the metalanguage. They are called *metalinguistic variables*. They perform logically just like the variables ('x', 'y', 'z') of elementary logic, except that they range over linguistic expressions in the object language.

7. Use versus Mention

In order to *use* a language (e.g., English) to discuss (*mention*) any domain of objects, the language of use should include names of those objects. This is based on the following principle.

> In order to *mention* an object, one *uses* a noun phrase that refers to that object;
> For example, one *uses* the name of that object.

For example, in order to *mention* former president Bush, one can *use* his name (although this is not the only way). Similarly, in order to *mention* our planet, we can *use* its name – 'The Earth'.

Observe that, if we follow the above prescribed use-mention custom, then the colloquial expression

I thought I heard my name *mentioned*

is technically incorrect, most of the time at least. The logically proper reply might be

actually, you heard your name *used*!

Of course, it might very well be the case that you did hear your name *mentioned*; for example, the persons involved were discussing how to spell your name.

8. Numerals versus Numbers

Arithmetic presents an excellent example of the distinction between use and mention. Arithmetic is a mathematical theory of (about) the natural numbers (counting numbers). [[See Chapter 2 for a formal/rigorous presentation of this theory.]]. In particular, arithmetic *uses* names of the various numbers to *mention* them. Now, the name of a number is usually called a *numeral*; if we want to *mention* a number, we *use* a numeral, or some other numerical expression, that refers to that number. This is summarized in the following principle.

> Numerals are names of numbers;
> we *use* numerals to *mention* numbers.

Now, there is exactly one system of natural numbers (supposing mathematical realism). On the other hand, there are many quite distinct numeral systems, including Egyptian, Hebrew, Greek, Roman, Arabic, etc. Furthermore, within the Arabic numeral system, currently adopted by most of the world, there are many numeral *sub*systems. – decimal, binary, etc.

Once we are clear about numbers versus numerals, we can immediately see that the following colloquial expressions are fallacious; in particular, they commit the error of *use-mention confusion*.

10 is a round number
100 is a round number
etc.

10 is a double-digit number (so 10% inflation is double-digit inflation)
100,000 is a six-figure number (so an income of $100,000 is a six-figure income)

What makes a number round? That it ends with a zero? But the number 10 doesn't end with a zero; its name does. We might as well say that the number 10 is mono-syllabic. Similarly with 'double-digit' and 'six-figure'. If we use the base-2 numeral system, then we almost always have "double-digit inflation", and most of us have at least "six-figure incomes".

In short, the above sentences are guilty of "linguistic voodoo" – ascribing a trait to an object based entirely on a feature of its name. The following are similar examples.

Otto is a round person;
Ohio is a round state;
New York, New Jersey, and New Mexico are all new states.

9. The Single-Quote Convention

Numerals name numbers; we *use* numerals to *mention* numbers. So how we *mention* numerals? For example, suppose I want to *mention* the (Arabic) numeral that denotes the number one. Notice that I cannot *use* the numeral itself; a numeral does not name itself; it names a number.

A standard (if not universal) custom is to take the numeral in question and enclose it within a pair of single quotes, and use the resulting expression to mention that numeral. This is an instance of a general principle.

> If one has a linguistic expression ε, then if one encloses ε in single quotes, then the resulting expression is a name of ε.

Let us consider a further simple example. If we want to mention former president Bush, we use *his* name, but if we want to mention *his* name, we use *its* name, which is obtained by taking *his* name and enclosing it in single quotes.

To see how this works, let us consider the following sentences.

(1)	Bush is a Republican	(true)
(2)	Bush is a 4-letter word	(false)
(3)	'Bush' is a Republican	(false)
(4)	'Bush' is a 4-letter word	(true)

In (1) and (2), I am using Bush's name to mention him; I am talking about the former president. On the other hand, in (3) and (4), I am not talking about the former president; I am talking about his (last) name.

We can also apply this technique to arithmetic and meta-arithmetic, as seen in the following examples.

(1)	42 is a number	(true)
(2)	42 is a 2-letter word	(false)
(3)	'42' is a number	(false)
(4)	'42' is a 2-letter word	(true)

Notice that the terms ‘letter’ and ‘word’ are expanded in English+ to encompass arithmetic expressions. This is sometimes clarified by introducing the term ‘alpha-numeric character’. Literally speaking, this is still not quite general enough, however, since we want ‘letter’ to encompass alpha characters, numeric characters, *and* punctuation marks.

The following are further examples to consider; each one is stated in question form [exercise].

(5)	is 7 a base-two number?	is ‘7’ a base-two number?
(6)	is 7 a base-two numeral?	is ‘7’ a base-two numeral?
(6)	is VII a number?	is ‘VII’ a number?
(6)	is 7 a Roman numeral?	is ‘7’ a Roman numeral?
(8)	is VII a Roman numeral?	is ‘VII’ a Roman numeral?

10. Off-Set Notation

Oftentimes – for example, in many of the numbered sentences above – we are not using a sentence but only mentioning it. This illustrates another very important linguistic convention.

> A linguistic expression can be mentioned by setting it off typographically.
> One off-set method is to set it on its own line(s).
> Another off-set method is to ornament it somehow – e.g., italics.
> In either case, the expression in question is *not used* but merely *displayed*.

Of course, context is exceedingly important! For example, the boxed expression directly above is not an instance of this technique. Sometimes a sentence is set off typographically, because it says something important or interesting. Also, the italic-method is not used in this way in the above boxed expression. The words in question are italicized, not for display, but for emphasis.

11. Elaborating on the Basic Single-Quote Scheme

Things get trickier. How do we mention the name of a name? One way, of course, is to set it off in display form. For example, whereas the following are examples of ordinary names,

Bush
10
100

the following are examples of names of names.

‘Bush’
‘10’
‘100’

What are the names of these expressions? Applying the single-quote naming scheme, we take the expressions in question, and surround them with single quotes, thus producing the following.

‘‘Bush’’
‘‘10’’
‘‘100’’

Indeed, if we take our single-quote naming scheme to its logical conclusion, we can produce an infinite list of names, as follows.

Bush
‘Bush’
‘‘Bush’’
‘‘‘Bush’’’
etc.

12. Single-Quotes versus Double-Quotes

Unfortunately, as it turns out, the twice-quoted expression

‘‘Bush’’

looks a lot like the double-quoted expression

“Bush”

so we have to be very careful reading such expressions. [Of course, the computer binary/hex representation of the above strings is not ambiguous.]

Double-quotation is yet another quotation technique. For example, many programming languages (e.g., Basic, Pascal, C++) use double-quotation as ***the*** method for referring to strings. Unfortunately, in English as variously practiced around the globe, single quotation and double quotation are each used in a variety of ways that are not entirely consistent.

Alas, there simply is no single universal consistent quotation convention in English. So, in order to make our own work consistent, precise, and rigorous, we must declare our own convention. This is described as follows.

(1) We use single quotes for naming linguistic expressions.
(2) We never use double quotation for this.
(3) We use double-quotes *primarily* for “raised-eyebrow” quotation.

Indeed, sentence (3) is both a statement about, and an illustration of, how we use double-quotation.

13. A Big Difference Between Single-Quotes and Double-Quotes

Having declared what quotation convention we are using, we next observe that there is a critical distinction between single quotes and double quotes, summarized as follows.

Single quotes are an integral part of the words they surround.

> Double quotes are not an integral part of the words they surround;
> rather, they are merely punctuation.

Let us elaborate on this. First, by punctuation, we mean those orthographic features of a sentence that are meant to organize and present it, including the usual punctuation symbols, plus capitalizing, boldfacing, italic, etc.

Internet Email gives us an example of punctuation not discussed by traditional grammar books – enclosing a word between asterisks – as in the following.

word

We can treat our use of double-quotation the same way. The critical fact is that asterisk-symbols – or double-quotes – flanking a word are not an integral part of the word, but part of the surrounding punctuation.

The following syntactic principle clarifies the notion of integral part.

> When one flanks a word with single-quotes, the resulting expression is a new, longer, word. More generally, when one flanks a string of symbols/characters with single-quotes, the result is a proper noun (of English+).

For example, the following is an *infinite* list of *distinct* words in (expanded) English.

Bush
‘Bush’
‘‘Bush’’
‘‘‘Bush’’’
‘‘‘‘Bush’’’’
etc.

Notice that each such word names the word above it (except for the first, which names the former U.S. president). Notice also that each word is two symbols longer than the previous word, so that the following are true sentences.

‘Bush’ is a 4-letter word;
‘‘Bush’’ is a 6-letter word;
‘‘‘Bush’’’ is an 8-letter word;
etc.

Here, we understand that sophisticated spelling in the expanded language English+ *must* include reference to single-quotes.

Single-quotation produces a new word everytime it is applied. What about double-quotation? The corresponding principle is the following.

> Flanking a word with double-quotes does not produce a new word, but is rather a special manner of presenting/displaying/using that word (not different in nature from bold-facing, italicizing, asterisk-flanking, etc.)

14. Another Example of Use-Mention

Generally, we do not get into trouble when we confuse use and mention. Consider the following perfectly acceptable bit of English.

the president's name is Bill Clinton

This seems perfectly understandable. However, suppose we combine this with the following identity statement.

Bill Clinton is the president.

Then we can use Identity Logic to obtain the following logical consequence.

the president's name is the president.

A valid argument has led to a false conclusion, so one of the premises must be false. The culprit is the first one. We understand it in a way that makes it true, but technically speaking it is false. The correct formulation of this statement is

the president's name is 'Bill Clinton'.

Interestingly enough, identity logic also provides the following conclusion.

Bill Clinton's name is 'Bill Clinton'

Indeed, we have here an instance of a "mostly valid" form. The following are other instances.

George Bush's name is 'George Bush';
Al Gore's name is 'Al Gore';
Ross Perot's name is 'Ross Perot'.

As it turns out, the general form of these sentences is difficult to write down. We will discuss problems like this in more detail later.

15. Symbolizing Meta-Linguistic Statements – 1

In elementary logic one learns to translate sentences from natural languages (e.g., English) into formal languages (e.g., Sentential Logic, Predicate Logic, etc.) The point of such translations is to reveal the logical forms of these sentences.

Now, one presumes that most of the sentences actually used in elementary logic books ought to submit to logical analysis. These include the syntactic rules of SL.

What happens when we apply elementary logic translation procedures to the syntactic rules of a formal language. Consider, for example, the formal language, originally described in Section 5. First its vocabulary is given as follows..

Vocabulary:

(1) upper case Roman letters:
A B C ……
(2) special symbols:
♯ ~ & ∨ → ↔ ( )
(3) nothing else.

Notice that these rules are lazily written, and should be rewritten as complete sentences [exercise]. The following are two those rewritten sentences

'P' is a symbol

'~' is a (special) symbol

These are fairly easy to translate, as follows. [See Section 26 on elementary logic translation schemes.]

Formula:	Translation Scheme:
S[p]	p : 'P' S[α] : α is a symbol
S[t]	t : '~' S[α] : α is a symbol

Clause (3) is an "extremal clause". It says that if a character is not mentioned in the list above it, then that character is not a symbol of the formal language in question. This is fairly difficult to symbolize, so we postpone it for the moment.

Next, let us consider the rules of formation.

Rules of Formation:

(a1) every upper case Roman letter is an atomic formula;
(a2) if 𝕊 is an atomic formula, then so is 𝕊♯;
(a3) nothing else is an atomic formula.

(f1) every atomic formula is a formula
(f2) if $\mathbb{S}$ is a formula, then so is:
$\sim\mathbb{S}$
(f3) if $\mathbb{S}_1$ and $\mathbb{S}_2$ are formulas, then so are:
$(\mathbb{S}_1 \,\&\, \mathbb{S}_2)$
$(\mathbb{S}_1 \vee \mathbb{S}_2)$
$(\mathbb{S}_1 \rightarrow \mathbb{S}_2)$
$(\mathbb{S}_1 \leftrightarrow \mathbb{S}_2)$
(f4) nothing else is a formula.

First, clause (a1) can be understood as short for 26 different clauses.

(1a) 'A' is an atomic formula
(1b) 'B' is an atomic formula
etc.

These are easy to symbolize.

Formula:	Translation Scheme:		
A[a]	a	:	'A'
A[b]	b	:	'B'
etc.	etc.		
	A[α]	:	α is an atomic formula

Clause (f1)

every atomic formula is a formula

is also an easy one

$\forall x(A[x] \rightarrow F[x])$	A[α]	:	α is an atomic formula
	F[α]	:	α is a formula

The extremal clauses

(a3) nothing else is an atomic formula
(f4) nothing else is a formula

are extremely difficult to symbolize. Indeed, we will spend an entire chapter (Chapter 3: Mathematical Induction) analyzing these kinds of sentences.

Between the really easy translations and the really hard translations are all the rest! Let us examine one of these.

if $\mathbb{S}$ is a formula, then so is $\sim\mathbb{S}$.

How do we symbolize this? First of all, it is implicitly universally quantified (a common technique in mathematics and logic). Here the variable of quantification is the funny-looking letter '$\mathbb{S}$', which is a metalinguistic variable ranging over strings of symbols. Making the quantifier and domain explicit, we rewrite it as follows.

for any **string** $\mathbb{S}$, if $\mathbb{S}$ is a formula, then so is $\sim\mathbb{S}$.

Concerning the term 'string' we have a choice: we can make strings of symbols the domain of quantification, or not. Let us do the former; accordingly, we do not have to symbolize '…is a string'. Putting what we have so far to work, we have the following *hybrid formula*, where we use (one of) the official variables of elementary logic.

$\forall x$ {if x is a formula, then so is $\sim x$}

To say that this is a hybrid formula is to say that it is only partially symbolized; there are components that must still be symbolized.

Next, the part inside the curly brackets is clearly equivalent to:

if x is a formula, then $\sim x$ is a formula

Using the following translation scheme,

$F[\alpha]$: α is a formula

yields the following.

(?) $\forall x\{F[x] \rightarrow F[\sim x]\}$

Important point: this is still a hybrid formula; it is not fully symbolized; it would not receive full credit on an elementary logic exam. Yes, it looks superficially like a true formula of elementary logic — but it isn't! The problem concerns '$\sim$'. This is a symbol of elementary logic, which formalizes the concept of negation, the official translation being 'it is not true that…'. Grammatically, negation is a one-place connective, which means that it takes a formula as input and generates a formula as output. In the above string of symbols '$\sim$' prefixes 'x', which is a variable, which is a noun phrase, not a formula.

Accordingly, from the viewpoint of elementary logic, the above string is not well-formed; it is gibberish.

16. The Role of the Tilde Symbol in English+

The problem at the end of the previous section prompts us to question exactly what is happening in our original sentence.

(t) if $\mathbb{S}$ is a formula, then so is $\sim\mathbb{S}$.

In particular, what is the role of the tilde symbol in sentence (t)? More generally, what is the role of the tilde symbol in the meta-language English+?

As we already know, (t) is not itself a formula of the object language $\mathcal{L}_S$; rather it is a sentence *in* the meta-language, English+, *about* the object language, $\mathcal{L}_S$. So, as a first approximation at least, whereas the object language $\mathcal{L}_S$ *uses* the tilde symbol (in particular, as a one-place connective), the metalanguage *mentions* it. So in particular, sentence (t) mentions the tilde symbol.

But wait! Recall our two fundamental principles.

Use-Mention Principle: In order to *mention* something, one *uses* its name.

> **Single-Quote Principle:** The name of a linguistic expression is obtained by enclosing that expression in single quotes.

Now the sentence (t) above presumably *mentions* the tilde symbol, so it should *use* the name of the tilde symbol. So why isn't the tilde symbol in quotes? Well, for one thing, that produces the following very odd expression.

(t′) if $\mathbb{S}$ is a formula, then so is '~'$\mathbb{S}$.

Obviously, the mere oddness of (t′) cannot by itself legitimize dropping the quotes. Also, if we drop the quotes here, shouldn't we also drop the quotes everywhere, in which case we have the following sorts of sentences.

~ is a symbol
P is an atomic formula

Of course, we did drop the quotes in the original formulation of the vocabulary of $\mathcal{L}_S$, but that formulation utilized off-set notation (recall Section 10). If we use in-line notation, we are supposed to use quotes.

17. A Conservative Formulation of English+

At this point, we should make a little more precise the grammatical rules of our metalanguage English+. As already mentioned, English+ is ordinary English augmented with various formal devices enabling it to serve as the metalanguage for formal languages such as $\mathcal{L}_S$. What are these formal devices?

First of all, English+ includes among its symbols metalinguistic variables,

$\mathcal{A}$, $\mathcal{B}$, C, $\mathbb{P}$, $\mathbb{Q}$, $\mathbb{R}$, etc. (with and without numerical subscripts)

which are variables that range over various expressions in $\mathcal{L}_S$.

Second, English+ includes among its symbols all the symbols of the object language,

P, Q, R, ~, &, etc.

Third, English+ has the following grammatical rule.

> If σ is a literal string of symbols of the object language, then the result of enclosing σ in single quotes is a proper noun.

[[Note that we have ascended to the meta-meta-language, English++, where the Greek letter 'σ' is one of *its* variables.]] So, for example, the following strings are all proper nouns of English+.

'~'
'P~'
'&~P'

At this point, if we provide no further rules for noun phrase generation, then we have a conservative formulation of English+. It is conservative in that the following strings are *not* proper nouns of English+.

~
P~
&~P

And indeed neither are the following.

~𝕊
(𝕊$_1$ & 𝕊$_2$)

This means that the following are ill-formed.

~ is a symbol
if 𝕊 is a formula, then so is ~𝕊
if S$_1$ and S$_2$ are formulas, then so is (𝕊$_1$ & 𝕊$_2$)

18. A More Liberal Formulation of the Grammar of English+

According to the conservative formulation of English+, the symbol '~' appears in both the object language – $\mathcal{L}_S$ – and the metalanguage – English+. On the other hand, whereas it appears as an autonomous grammatical entity in $\mathcal{L}_S$, it appears *merely* as a symbol in English+. In English+, a tilde by itself simply is not grammatical; in order to make a genuine grammatical entity, we employ the single-quote technique, which produces a proper noun of English+.

There is an alternative, more liberal, formulation of the grammar of English+. according to which the tilde symbol is not just a symbol of English+, it is a genuine proper noun of English+. The following is the proposed additional rule that yields this result.

> If σ is a symbol of the object language $\mathcal{L}$, then (no matter how σ is used in $\mathcal{L}$) the symbol σ itself is a proper noun in the meta-language English+. Furthermore, as used in English+, σ denotes itself.

According to this construal of English+, the tilde symbol in English+ is used to mention the tilde symbol in the object language. One symbol appears *grammatically* in both languages, $\mathcal{L}_S$ and English+, although it is used differently in the two languages. In particular, whereas tilde is a one-place connective in $\mathcal{L}_S$, it is a proper noun in English+.

In other words, according to this analysis, the tilde symbol is *ambiguous* between the two languages. This sort of ambiguity is not uncommon; the same string of characters can appear in different languages, and be used quite differently. For example, English and German both use the words 'boot' and 'also', but in quite different ways. Similarly, tilde can occur as a statement connective in SL and as a proper noun, naming that connective, in English+.

Given the more liberal construal of proper nouns in English+, we now see that the following is (or at least could be) perfectly grammatical.

~ is a symbol

Generally, a name does not resemble its bearer; e.g., 'Bush' does not resemble Bush, even remotely, although ''Bush'' resembles 'Bush'. In the metalanguage of logic, however, it is sometimes convenient to use names that look just like their bearers; hence the use of symbols *ambiguous* between the object language and the metalanguage.

19. Other Approaches to Symbolic Use versus Mention

We have now considered two methods of mentioning object language symbols – the single-quote method, and double-use method. A method, intermediate between these two, the ornamental method, is to use different fonts; for example, one can make the object language symbol lighter in weight than its metalanguage name. Technically, we have two different symbols, so we don't have an explicit ambiguity, but we do have that the name of a symbol strongly resembles the symbol.

The most radical, but simplest, approach to naming symbols is simply never to display the object language in the first place! In this case, every logical symbol (tilde, ampersand, etc.) is not the actual logical symbol, but only its name. The actual negation sign, for example, is never displayed; its actual orthographic nature is left completely unspecified.

When you think about it, most disciplines are like this. For example, arithmetic never actually displays the numbers, but only their names (the numerals). Similarly, geography never displays the continents, oceans, etc., but only symbols of them (including maps). For obvious reasons, geography is not a show-and-tell discipline.

If we use the same approach to metalogic, we talk *about* the object language symbols, but we never actually *show* them. In this case, the tilde symbol is not ambiguous; it is used exactly one way – as a proper noun to denote the negation sign of the object language, whatever that sign actually is. The actual negation sign – in other words, the actual glyph – may or may not look anything like its name. But then, why should it?

The following table summarizes the four approaches.

Approach	Symbol	Category of Symbol in Object Language	Category of Symbol in Meta-Language	Symbol's name
single-quote	~	one-place connective	none	'~'
double-use	~	one-place connective	proper noun	~
ornamental	~	one-place connective	none	*~*
no display	???	one-place connective	???	~

Here, the asterisks mean that the symbol is ornamented in some manner, for example, by boldfacing; furthermore, it is understood that *in this particular case* the ornamentation produces a new symbol rather than a different presentation of the same symbol.

20. Symbolizing Meta-Linguistic Statements – 2

Let us now return to the task of symbolizing the statements used to describe the syntax of SL. Originally we stumbled on the following sentence.

(t) if $\mathbb{S}$ is a formula, then so is $\sim\mathbb{S}$

which we had partially symbolized as follows.

(?) $\forall x\{F[x] \rightarrow F[\sim x]\}$

According to our current analysis, the tilde symbol as used in the metalanguage English+ may be understood as a proper noun. In elementary logic, we use lower case Roman letters to symbolize proper nouns. The following is a plausible symbolization.

t : $\sim$

Accordingly, we obtain the following expression.

$\forall x\{F[x] \rightarrow F[tx]\}$

This is only slightly better. Although it is not complete gibberish, it is still not well-formed according to the syntactic canons of elementary logic. The difficulty is the expression 'tx'.

What is the grammatical status of 'tx'? Well, we know that 't' is a proper noun, and 'x' is a variable, so they are both noun phrases. So the expression 'tx' consists of two noun phrases slammed together, treated as a compound noun phrase, which is then inserted as a grammatical unit into a one-place predicate.

There is a familiar natural language counterpart of this process. Often, in elementary algebra, we take two nouns – e.g.,

2

and

x

and concatenate them to produce the compound noun phrase

2x

then go on to form sentences such as

y=2x

It is understood, of course, that

2x

is short for

2 times x

So when we translate the formula ‘y=2x’ into symbolic logic, we must include a symbolization of ‘times’. This requires the logic of function signs (Function Logic). The translation is then given as follows.

E[y, t(2,x)]

Here, the supporting lexicon is:

E[α,β]	:	α equals β
t(α,β)	:	α times β
2	:	two

Notice that we adopt a useful and natural simplification that treats the (Arabic decimal) numerals as an acceptable part of the symbolization. Notice also that we have only translated the sentence into Function Logic. We can take the translation one step further, by using Identity Logic. In that case, ‘equals’ gets a special logical symbol ‘=’, written in its natural position. This yields the following formula.

y = t(2,x)

Noun juxtaposition can lead to problems. Although it is proper and natural to read ‘2x’ as short for ‘2 times x’, it would be improper to read ‘22’ as ‘2 times 2’. Unfortunately, arithmetic employs two different juxtaposition schemes – algebraic and decimal – but they are in serious conflict with each other!

Now, back to our original sentence.

if 𝕊 is a formula, then so is ∼𝕊

It is apparent that noun-juxtaposition is also at work here. We have a complex noun phrase ‘∼𝕊’, which is obtained by concatenating two simple nouns ‘∼’ and ‘𝕊’. If we are careful reading this sentence “out loud”, we see what is happening more clearly.

> if 𝕊 is a formula, then the result of concatenating ‘∼’ with 𝕊 (in that order) is also a formula.

The symbolization of this is

∀x{F[x] → F[c(t,x)]}

where the supporting lexicon is:

F[α]	:	α is a formula
c(α,β)	:	the result of concatenating α and β (in that order)
t	:	‘∼’

21. String Addition

Bare (implicit) juxtaposition is not the only way of formulating juxtaposition in English+. Another, more grammatically revealing, technique introduces an explicit symbol for juxtaposition. For example, one can introduce a 2-place string-addition operator, ‘+’, written in infix notation, so that the expression

s+t

refers to the result of appending string t to string s. For example,

‘y’ + ‘ear’ = ‘year’
‘ear’ + ‘ly’ = ‘early’

The plus-sign is not a completely capricious choice. First, the original Basic programming language uses ‘+’ in this manner (although the more contemporary Visual Basic uses ampersand instead). Second, the resulting algebra of strings (appropriately defined) is a semi-group with identity (a “monoid”). In particular, the string-addition operation is associative, but not commutative. So the order of addition matters, but not the grouping, so parentheses can be dropped.

Using the string addition operation, we can rewrite (t) *schematically* as follows.

(t+) if $\mathbb{S}$ is a formula of SL, then so is $\tau+\mathbb{S}$

This is schematic; the Greek letter ‘τ’ stands temporarily in place of the *name* of the tilde symbol, whatever that happens to be – tilde, quoted tilde, bolded tilde, or something else entirely. The string addition notation works with any of the tilde-naming methods.

String addition, so defined, is a two-place operation, but since it is associative [i.e., a+(b+c) = (a+b)+c], one can drop parentheses, thus producing the following sorts of equations.

‘c’ + ‘a’ + ‘t’ =
‘c’ + ‘at’ =
‘ca’ + ‘t’ = ‘cat’

Also, since the plus-sign is associative, we can naturally define associated 3-place, 4-place operations, etc., as well.

(a2) $\Sigma(a,b) =_{df} a+b$
(a3) $\Sigma(a,b,c) =_{df} \Sigma(a,b) + c$
(a4) $\Sigma(a,b,c,d) =_{df} \Sigma(a,b,c) + d$
etc.

If we insist on a *standard* first-order formalization of the metalanguage, then ‘Σ’ is ambiguously used as a 2-place function sign, a 3-place function sign, etc. If we loosen our formal standards, we can regard ‘Σ’ as a single *anadic* function sign. An anadic function sign is one that takes an arbitrary finite number of singular terms to produce a singular term. [[Anadic logic is expressively richer but deductively poorer; there’s a trade off]].

22. Quine Quotes

Anadic string addition is intimately related to a well-known, and often used, metalogical method of quotation, due originally to Quine. According to this method, which uses a special category of quotes, called *Quine quotes* or *corner quotes*, (t) is rewritten as follows.

(tq) if $\mathbb{S}$ is a formula of SL, then so is $\ulcorner\sim\mathbb{S}\urcorner$

This is the standard Quinean way to rewrite (t), irrespective of which tilde-naming convention we adopt. Both of the following supposedly make sense.

if $\mathbb{S}$ is a formula of SL, then one can prefix $\mathbb{S}$ with ‘$\sim$’, and the result is $\ulcorner\sim\mathbb{S}\urcorner$

if $\mathbb{S}$ is a formula of SL, then one can prefix $\mathbb{S}$ with $\sim$, and the result is $\ulcorner\sim\mathbb{S}\urcorner$

Notice, in particular that, if quotes are used to make tilde's name, they are dropped when inside Quine quotes.

Apart from the dropping of "extra" quotes, and the absence of commas, Quine-quotation is formally identical to anadic string addition. The following are three metalinguistic sentences about double-negation; one uses anadic string addition; the second one uses Quine quotes; the last one uses binary string addition (without parentheses).

(a) if $\mathbb{S}$ is a formula, then so is $\Sigma(\text{`}\sim\text{'}, \text{`}\sim\text{'}, \mathbb{S})$;

(q) if $\mathbb{S}$ is a formula, then so is $\ulcorner\sim\sim\mathbb{S}\urcorner$;

(b) if $\mathbb{S}$ is a formula, then so $\text{`}\sim\text{'} + \text{`}\sim\text{'} + \mathbb{S}$.

Here, we use the single-quote method for naming the tilde symbol.

23. A Formal Account of Quine-Quotation

How exactly do Quine-quotes work? For example, which of the following expressions are well-formed expressions of the meta-language.

$\ulcorner\mathcal{A}\urcorner$
$\ulcorner\sim\mathcal{A}\urcorner$
$\ulcorner\sim\urcorner$
$\ulcorner\text{`}\sim\text{'}\urcorner$
$\ulcorner\mathrm{P}\urcorner$
$\ulcorner\text{`}\mathrm{P}\text{'}\urcorner$
$\text{`}\ulcorner\mathrm{P}\urcorner\text{'}$
$\ulcorner\ulcorner\mathrm{P}\urcorner\urcorner$
$\ulcorner\ulcorner\mathcal{A}\urcorner\urcorner$
$\ulcorner\text{`}\mathcal{A}\text{'}\urcorner$
$\ulcorner\sim\mathcal{A}\sim\urcorner$

In order to answer this question definitively, we must at least partially formalize the meta-language. Toward that end, we offer the following rules of formation.

1. Every symbol of the object language $\mathcal{L}$ is also a symbol of the meta-language $\mathcal{ML}$; this set is called Sym($\mathcal{L}$).
2. Left-single-quote and right-single quote are symbols of $\mathcal{ML}$, *but not* $\mathcal{L}$.
3. Quine-quotes are symbols of $\mathcal{ML}$, *but not* $\mathcal{L}$.
4. Symbols of $\mathcal{ML}$ include meta-linguistic variables [e.g., '$\mathcal{A}$', '$\mathcal{B}$', etc.] that range over expressions of $\mathcal{L}$; this set is called Var($\mathcal{ML}$). These are *not* symbols of $\mathcal{L}$.
5. Quotable strings:
   Any string of symbols from Sym($\mathcal{L}$) is a quotable string; nothing else is.
6. Quote expressions:
   Any quotable string flanked by ordinary quotes is a quote expression; nothing else is;
   every such expression is a proper noun of $\mathcal{ML}$.

7. Quine-quotable strings:
   Any string of symbols from Sym($\mathcal{L}$) $\cup$ Var($\mathcal{ML}$) is a Quine-quotable string; nothing else is.

8. Quine-quote expressions:
   Any Quine-quotable string flanked by Quine-quotes is a Quine-quote expression; nothing else is; every such expression is a noun phrase of $\mathcal{ML}$.

With these rules in hand, we can answer the question concerning which expressions above are well-formed.

⌜$\mathcal{A}$⌝	:	well-formed
⌜~$\mathcal{A}$⌝	:	well-formed
⌜~⌝	:	well-formed
⌜‘~’⌝	:	ill-formed
⌜P⌝	:	well-formed
⌜‘P’⌝	:	ill-formed
‘⌜P⌝’	:	ill-formed
⌜⌜P⌝⌝	:	ill-formed
⌜⌜$\mathcal{A}$⌝⌝	:	ill-formed
⌜‘$\mathcal{A}$’⌝	:	ill-formed
⌜~$\mathcal{A}$~⌝	:	well-formed

24. Dropping Quine Quotes Altogether

Now, Quine quotation is clearly notationally more economical than string addition. However, it is not the most economical system available. Really, the only thing that Quine quotes do is remind us that the expression in question is metalinguistic. If we are clear about our notational conventions, then Quine quotes are unnecessary.

If we remove the Quine quotes, we obtain one of the following, depending on whether tilde's metalinguistic name involves quotes or not.

(m1) if $\mathbb{S}$ is a formula, then so is ~$\mathbb{S}$
(m2) if $\mathbb{S}$ is a formula, then so is ‘~’$\mathbb{S}$

These two sentences are perfectly O.K., so long as we adopt the following metalinguistic convention.

> if μ_1 and μ_2 are metalinguistic noun phrases (simple or complex), which refer to object language expressions ε_1 and ε_2, respectively, then the *bare juxtaposition* of μ_1 and μ_2 is, *by convention*, a complex noun phrase that refers to the juxtaposition of the expressions ε_1 and ε_2.

In other words, we simply drop the plus sign, and make it implicit in the notation. As we have already noted, this is analogous to arithmetic, where one represents the multiplication operation simply by juxtaposition as in the formula ‘a(b+c)=ab+ac’.

25. Summary: Approaches to Metalinguistic Singular Terms

In conclusion, let us review the different approaches to metalinguistic singular terms. We must deal with two kinds of metalinguistic singular terms.

(1) names of atomic symbols of the object language;

(2) names of molecular symbols (strings of symbols) of the object language.

In each case, we discuss four approaches.

1. 4 Ways to deal with atomic symbols

(1) **double-use method:**
use one symbol *ambiguously*, both in the object language, and in the metalanguage;

(2) **ornamental method:**
use two symbols, but which look very similar; for example, use tilde in the object language, and use bold-faced tilde in the metalanguage;

(3) **single-quote method:**
use single quotes to distinguish the symbol from its name;

(4) **no-display method:**
use one symbol, but exclusively in the metalanguage, to name the object language symbol; do not explicitly display the object language symbol, leaving its actual orthographic nature (glyph) completely unspecified.

2. 4 Ways to deal with molecular symbols

(1) binary string addition: ..+..;

(2) anadic string addition: Σ(..,..,..);

(3) Quine quotation: ⌜...⌝;

(4) implicit (bare) juxtaposition.

Each approach to molecular symbols is consistent with each approach to atomic symbols. Which combination of methods we choose is largely a matter of taste; of course, we should try to be consistent in any given context.

26. Appendix: General Translation Scheme for Elementary Logic

In this appendix we briefly discuss the salient features of elementary logic translation.

First, only a few simple grammatical categories are considered in first-order logic. [See Chapter 2 for explanation of the notion of *first-order*.] These include the following.

sentences	S
noun phrases	N
predicates	Nk➔S
function signs	Nk➔N
connectives	Sk➔S
quantifiers	V+S➔S
description operator	V+S➔N

See Appendix 3 for a more thorough description of categorial grammar.

In symbolizing a sentence 𝕊 of English, one first identifies the grammatical components of 𝕊, the smallest of which are then abbreviated by Roman letters (plus category information), after which one translates 𝕊 in a systematic manner.

Example 1:

Jay is a Freshman

First, the abbreviation scheme is given as follows.

j	:	Jay	proper noun
F[α]	:	α is a Freshman	1-place predicate

Note that the predicate symbolization includes category information; in particular, the square brackets indicate that it is a predicate; also, the single argument position is *schematically* filled by the Greek letter 'α'. The final symbolization is:

F[j]	sentence

Example 2:

Jay respects Kay

j	:	Jay	proper noun
k	:	Kay	proper noun
R[α,β]	:	α respects β	2-place predicate

R[j,k]	sentence

So far we have only dealt with Predicate Logic translations. We also must consider Function Logic, which adds function signs, which are grammatical functors that take noun phrases and generate noun phrases. As with predicates, we use lower case Greek letters as schematic variables.

Example 3:

Jay's mother respects Kay's mother

First, let us "handicap" ourselves and translate this sentence into Predicate Logic. This produces the following translation.

m_1	:	Jay's mother	proper noun(?)
m_2	:	Kay's mother	proper noun(?)
$R[\alpha,\beta]$	:	α respects β	2-place predicate
$R[m_1,m_2]$			sentence

This is grammatically, and logically, inadequate. The problem is that Predicate Logic has no means of dealing with compound noun phrases. Enter Function Logic, which provides the following symbolization.

Jay's mother respects Kay's mother

j	:	Jay	proper noun
k	:	Kay	proper noun
$m(\alpha)$	:	α's mother	function sign
$R[\alpha,\beta]$	:	α respects β	2-place predicate
R[m(j),m(k)]			sentence

Note that, for function signs, we use lower case Roman letters, and we use round parentheses for further categorial delineation.

1. Summary of Translation Scheme:

Proper nouns: lower case Roman letters: a, b, c, … (with or without subscripts)

examples:	j	:	Jay
	s	:	the Sears Tower
	2	:	two

Predicates: upper case Roman letters, plus square brackets:

examples:	$H[\alpha]$	:	α is happy
	$R[\alpha,\beta]$	:	α respects β
	$S[\alpha,\beta,\gamma]$	:	α sold β to γ
	$H[\alpha,\beta,\gamma,\delta]$	:	α is happy that β sold γ to δ

Function Signs: lower case Roman letters, plus parentheses:

examples:	$m(\alpha)$	:	the mother of α
	$s(\alpha,\beta)$	:	the sum of α and β
	$s(\alpha,\beta,\gamma)$	:	the sum of α, β, γ

Variables: lower case Roman letters: z, y, x, …

27. Exercises

#1. Symbolize the following using the language of elementary first-order logic. See Section 26 for general symbolization scheme.

1. Bush's name is 'Bush'.
2. Although Bush can spell 'Bush', he cannot spell ''Bush''.
3. The first letter of Bush's name is 'B'.
4. The first letter of the name of 'Bush' is ' '.
5. The word ''Bush'' is used to mention the word 'Bush'.
6. If one takes an expression and encloses it in single quotes, then the result is a name of that expression. [ignore the anthropocentric nature of this sentence].

#2. For each "Bush-sentence" in #1, write down a true sentence concerning which occurrences of
Bush
'Bush'
''Bush''
etc.
are *mentioned*, and which occurrences are *used categorially* (grammatically), and which are used but *not categorially*. Note, in this connection, that the word 'bush' occurs inside the larger word 'bushel' but also inside the larger word ''bush'' — but *not categorially*.

#3. Given the following metalinguistic identities,

$\mathcal{A}$ = 'P'
$\mathcal{B}$ = '~Q'

complete each of the following metalinguistic equations to form a true and directly informative sentence using quote-only notation, **unless** it is ill-formed in the meta-language, in which case write 'ill-formed'. [Example, if $\mathcal{A}$ = 'P', then $\ulcorner\mathcal{A}\rightarrow\mathcal{A}\urcorner$ = 'P→P'.]

1. ‚$\ulcorner\sim\mathcal{A}\urcorner$ =
2. $\sim\ulcorner\mathcal{A}\urcorner$ =
3. ‚$\ulcorner\sim\mathcal{B}\urcorner$ =
4. ‚$\ulcorner\sim(\mathcal{A}\rightarrow\mathcal{B})\urcorner$ =
5. ‚$\ulcorner\mathcal{A}\urcorner\rightarrow\ulcorner\mathcal{B}\urcorner$ =
6. $\ulcorner\mathcal{A}\rightarrow(\mathcal{B}\rightarrow\mathcal{A})\urcorner$ =
7. $\ulcorner\mathcal{A}\sim\mathcal{B}\urcorner$ =

#4. The following is an example of a formal language (written somewhat sloppily, perhaps).

1. Vocabulary: ‘p’, ‘♯’, ‘N’, ‘K’.

2. Rules of Formation:
   Atomic Formulas:
   - a1. ‘p’ is an atomic formula;
   - a2. if σ is an atomic formula, then so is σ♯.
   - a3. nothing else is an atomic formula.

   Formulas:
   - f1. Every atomic formula is a formula;
   - f2. if σ is a formula, then so is: Nσ;
   - f3. if σ_1 and σ_2 are formulas, then so is: K$\sigma_1\sigma_2$;
   - f4. nothing else is a formula.

Rewrite this syntax, first using Quine-quote notation, then using quote-plus notation.

#5. Symbolize the sentences in Problem #4 in the language of elementary logic. Make sure that all implicit logical operators (quantifiers, functors, etc.) are made explicit. Be sure to write down your lexicon with category information, and be sure to identify domain of quantification.

#6. In reference to the formal language specified in Problem #4, the following are all true statements [where we use quote-notation].

1. ‘p♯’ is a formula.
2. ‘Kp♯Kp♯♯Np’ is a formula.
3. ‘♯p’ is not an atomic formula.
4. every atomic formula begins with the symbol ‘p’.
5. no molecular formula begins with the symbol ‘p’.
6. every molecular formula begins with ‘K’ or ‘N’.
7. every formula begins with ‘p’, ‘K’, or ‘N’

For each of these, give a formal proof. In some cases, the proof is impossible given the resources currently at our disposal. Still… think about it!

28. Answers to Exercises

#1.

1. n(b) = a
2. S[b,a] & ~S[b,c]
3. f(n(b)) = e
4. f(n(a)) = q
5. M[c, a]
6. $\forall x\{E[x] \rightarrow N[q(x), x]\}$

Lexicon:

n(α)	:	α's name
b	:	Bush
a	:	'Bush'
c	:	''Bush''
e	:	'B'
q	:	' '
S[α,β]	:	α can spell β
M[α,β]	:	α is used to mention β
E[α]	:	α is an expression
N[α,β]	:	α is a name of β
q(α)	:	the result of enclosing α in single quotes

#2.

1. Bush is mentioned, but not used;
   'Bush' is used and mentioned;
   ''Bush'' is used but not mentioned;
   the rest are neither used nor mentioned.

2. Bush is mentioned, but not used;
   'Bush' is used and mentioned;
   ''Bush'' is used and mentioned;
   '''Bush''' is used but not mentioned;
   the rest are neither used nor mentioned.

3. Bush is mentioned, but not used;
   'Bush' is used but not directly mentioned,
   although it is indirectly mentioned [by 'Bush's name'];
   the rest are neither used nor mentioned.

4. Bush is neither mentioned nor used;
   'Bush' is used (but not categorially) and mentioned;
   ''Bush'' is used but not directly mentioned,
   although it is indirectly mentioned [' the name of 'Bush' '];
   the rest are neither used nor mentioned.

5. Bush is neither mentioned nor used;
'Bush' is mentioned; it is also used but *not* categorially
(similarly 'u' is used but not categorially)
''Bush'' is both used and mentioned;
'''Bush''' is used but not mentioned;
the rest are neither used nor mentioned.

#3.

1. ⌜$\sim\mathcal{A}$⌝ = '$\sim$P'
2. $\sim$⌜$\mathcal{A}$⌝ = ill-formed
3. ⌜$\sim\mathcal{B}$⌝ = '$\sim\sim$Q'
4. ⌜$\sim(\mathcal{A}\rightarrow\mathcal{B})$⌝ = '$\sim$(P$\rightarrow\sim$Q)'
5. ⌜$\mathcal{A}$⌝$\rightarrow$⌜$\mathcal{B}$⌝ = ill-formed
6. ⌜$\mathcal{A}\rightarrow(\mathcal{B}\rightarrow\mathcal{A})$⌝ = 'P$\rightarrow$($\sim$Q$\rightarrow$P)'
7. ⌜$\mathcal{A}\sim\mathcal{B}$⌝ = 'P$\sim\sim$Q'

#4.

Using Quine-Quotation:

1. Vocabulary: 'p', '#', 'N', 'K'.
2. Rules of Formation:
   Atomic Formulas:
   - a1. 'p' is an atomic formula;
   - a2. if σ is an atomic formula, then so is ⌜σ#⌝;
   - a3. nothing else is an atomic formula.

   Formulas:
   - f1. Every atomic formula is a formula;
   - f2. if σ is a formula, then so is: ⌜Nσ⌝;
   - f3. if σ_1 and σ_2 are formulas, then so is: ⌜K$\sigma_1\sigma_2$⌝;
   - f4. nothing else is a formula.

Using plus-quote notation:

1. Vocabulary: 'p', '#', 'N', 'K'.
2. Rules of Formation:
   Atomic Formulas:
   - a1. 'p' is an atomic formula;
   - a2. if σ is an atomic formula, then so is σ+'#'.
   - a3. nothing else is an atomic formula.

   Formulas:
   - f1. Every atomic formula is a formula;
   - f2. if σ is a formula, then so is: 'N'+σ
   - f3. if σ_1 and σ_2 are formulas, then so is: 'K'+σ_1+σ_2;
   - f4. nothing else is a formula.

#5.

1.1. A[p]
1.2. $\forall x(A[x] \rightarrow A[j(x,s)])$
1.3. this is very tricky; see later chapter on mathematical induction
2.1. $\forall x(A[x] \rightarrow F[x])$
2.2. $\forall x(F[x] \rightarrow F[j(n,x)])$
2.3. $\forall x \forall y(F[x] \,\&\, F[y] .\rightarrow F[j(k,x,y)])$
2.4. see 1.3 above.

Lexicon:
Domain of quantification: symbols and strings of symbols.

p	:	'p'	A[α]	:	α is an atomic formula
s	:	'#'	F[α]	:	α is a formula
k	:	'K'			
n	:	'N'			
j(α_1,α_2...)	:	the result of juxtaposing α_1 in front of α_2 in front of ... (anadic)			

#6.1

Formal Proof:

(1)	~~SHOW~~: 'p#' is a formula	DD
(2)	| 'p' is an atomic formula	a1
(3)	| 'p#' is an atomic formula	2,a2,QL
(4)	| 'p#' is a formula	3,f1,QL

Informal Proof.

By a1, 'p' is an atomic formula; so by a2, 'p#' is an atomic formula; so by f1, 'p#' is a formula (QED).

#6.2

Formal Proof:

(1)	~~SHOW~~: 'Kp#Kp##Np' is a formula	DD
(2)	| 'p' is an atomic formula	a1
(3)	| 'p#' is an atomic formula	2,a2,QL
(4)	| 'p##' is an atomic formula	3,a2,QL
(5)	| 'p##' and 'p' are formulas	2,4,f1,QL
(6)	| 'Kp##p' is a formula	5,f3,QL
(7)	| 'p' is a formula	2,f1,QL
(8)	| 'Np' is a formula	7,f2,QL
(9)	| 'KpKp##p' is a formula	6,8,f3,QL

Informal Proof.

By a1, 'p' is an atomic formula; so by a2, 'p#' is an atomic formula; so by a2, 'p##' is a an atomic formula. Since 'p##' and 'p' are atomic formula, by f1 they are also formulas. But then by f3, 'Kp##p' is a formula. But also, since 'p' is a formula, by f2, 'Np' is a formula. Finally, since 'Kp##p' and 'Np' are formulas, by f3, 'KpKp##Np' is a formula (QED).

#6.3

Very Informal Proof:

(1)	Every atomic formula begins with the letter 'p'	Lemma 1
(2)	'♯p' does not begin with the letter 'p'	Lemma 2
(3)	therefore, '♯p' is not an atomic formula	1,2,QL

What is a lemma? First, a *theorem* is a proposition/statement that can be deduced from the principles "on hand". For example, if we are proving something about a specific formal language $\mathcal{L}$, then the formal rules of $\mathcal{L}$ count as "on hand". A *lemma* is a species of theorem, although the distinction is pragmatic, not theoretical. A lemma is simply a subordinate theorem of one or more larger theorems.

To make the above informal proof more formal, we need to formalize the notion of *begins with*, which requires a formal deductive theory of strings, which we do in a later Chapter. Given this formal theory, it is easy to show *logically* that '♯p' does not begin with 'p'. That takes care of Lemma 2. Lemma 1 is not so easy. It requires a formal elucidation of the extremal clause 'nothing else is an atomic formula'. This will wait until Chapter 3 (Mathematical Induction). By way of preview, the very informal argument will go as follows.

(1)	'♯p' ≠ 'p'	follows from Lemma 2
(2)	if '♯p' ≠ σ, then '♯p' ≠ σ+'♯'	must be proven using theory of strings
(3)	'♯p' ≠ *any* atomic formula	1,2 + math induction*
(4)	'♯p' is not an atomic formula	3,IL

*General mathematical induction constitutes the formal elucidation of all extremal clauses.

6.4 – 6.7

These require mathematical induction, so we postpone the solutions to these.